**Asp.Net II Mid Term 2018 Solution set -A**

**Q.1 Rich Controls**

ASP.NET provides large set of controls. These controls are divided into different categories, depends upon their functionalities. The followings control comes under the rich controls category.

* FileUpload control
* Calendar control
* AdRotator control
* MultiView control
* Wizard control

#### FileUpload control

FileUpload control is used to browse and upload files. After the file is uploaded, you can store the file on any drive or database. FileUpload control is the combination of a browse button and a text box for entering the filename.

**The FileUpload control supports the following important properties.**

* **FileBytes:** It returns the contents of uploaded file as a byte array
* **FileContent:** You can get the uploaded file contents as a stream.
* **FileName:** Provides the name of uploaded file.
* **HasFile:** It is a Boolean property that checks whether particular file is available or not.
* **PostedFile:** Gets the uploaded file wrapped in the HttpPostedFile object.

2. The **base keyword** is used to access members of the base class from within a derived class:

* Call a method on the base class that has been overridden by another method.
* Specify which base-class constructor should be called when creating instances of the derived class.

A base class access is permitted only in a constructor, an instance method, or an instance property accessor.

It is an error to use the base keyword from within a static method.

In this example, both the base class, Person, and the derived class, Employee, have a method named Getinfo. By using the base keyword, it is possible to call the Getinfo method on the base class, from within the derived class.

C#

public class Person
{
 protected string ssn = "444-55-6666";
 protected string name = "John L. Malgraine";

 public virtual void GetInfo()
 {
 Console.WriteLine("Name: {0}", name);
 Console.WriteLine("SSN: {0}", ssn);
 }
}
class Employee : Person
{
 public string id = "ABC567EFG";
 public override void GetInfo()
 {
 // Calling the base class GetInfo method:
 base.GetInfo();
 Console.WriteLine("Employee ID: {0}", id);
 }
}

class TestClass
{
 static void Main()
 {
 Employee E = new Employee();
 E.GetInfo();
 }
}
/\*
Output
Name: John L. Malgraine
SSN: 444-55-6666
Employee ID: ABC567EFG

3. **Verification:** This verifies whether the product is done according to the specification? The process of determining whether of not the products of a given phase of the software development cycle meet the implementation steps and can be traced to the incoming objectives established during the previous phase. The techniques for verification are testing, inspection and reviewing.

**Validation:** This checks whether the product meets the customer requirement? The process of evaluating software at the end of the software development process to ensure compliance with software requirements. The techniques for validation is testing, inspection and reviewing.

4. **Ajax control in asp.net application:**

AJAX stands for Asynchronous JavaScript and XML. This is a cross platform technology which speeds up response time. The AJAX server controls add script to the page which is executed and processed by the browser.

However like other ASP.NET server controls, these AJAX server controls also can have methods and event handlers associated with them, which are processed on the server side.

The control toolbox in the Visual Studio IDE contains a group of controls called the 'AJAX Extensions'



## The ScriptManager Control

The ScriptManager control is the most important control and must be present on the page for other controls to work.

It has the basic syntax:

<asp:ScriptManager ID="ScriptManager1" runat="server">

</asp:ScriptManager>

If you create an 'Ajax Enabled site' or add an 'AJAX Web Form' from the 'Add Item' dialog box, the web form automatically contains the script manager control. The ScriptManager control takes care of the client-side script for all the server side controls.

## The UpdatePanel Control

The UpdatePanel control is a container control and derives from the Control class. It acts as a container for the child controls within it and does not have its own interface. When a control inside it triggers a post back, the UpdatePanel intervenes to initiate the post asynchronously and update just that portion of the page.

For example, if a button control is inside the update panel and it is clicked, only the controls within the update panel will be affected, the controls on the other parts of the page will not be affected. This is called the partial post back or the asynchronous post back.

### Example

Add an AJAX web form in your application. It contains the script manager control by default. Insert an update panel. Place a button control along with a label control within the update panel control. Place another set of button and label outside the panel.

The design view looks as follows:



The source file is as follows:

<form id="form1" runat="server">

 <div>

 <asp:ScriptManager ID="ScriptManager1" runat="server" />

 </div>

 <asp:UpdatePanel ID="UpdatePanel1" runat="server">

 <ContentTemplate>

 <asp:Button ID="btnpartial" runat="server" onclick="btnpartial\_Click" Text="Partial PostBack"/>

 <br />

 <br />

 <asp:Label ID="lblpartial" runat="server"></asp:Label>

 </ContentTemplate>

 </asp:UpdatePanel>

 <p> </p>

 <p>Outside the Update Panel</p>

 <p>

 <asp:Button ID="btntotal" runat="server" onclick="btntotal\_Click" Text="Total PostBack" />

 </p>

 <asp:Label ID="lbltotal" runat="server"></asp:Label>

</form>

Both the button controls have same code for the event handler:

string time = DateTime.Now.ToLongTimeString();

lblpartial.Text = "Showing time from panel" + time;

lbltotal.Text = "Showing time from outside" + time;

Observe that when the page is executed, if the total post back button is clicked, it updates time in both the labels but if the partial post back button is clicked, it only updates the label within the update panel.



A page can contain multiple update panels with each panel containing other controls like a grid and displaying different part of data.

When a total post back occurs, the update panel content is updated by default. This default mode could be changed by changing the UpdateMode property of the control. Let us look at other properties of the update panel.

### Properties of the UpdatePanel Control

The following table shows the properties of the update panel control:

|  |  |
| --- | --- |
| **Properties** | **Description** |
| ChildrenAsTriggers | This property indicates whether the post backs are coming from the child controls, which cause the update panel to refresh. |
| ContentTemplate | It is the content template and defines what appears in the update panel when it is rendered. |
| ContentTemplateContainer | Retrieves the dynamically created template container object and used for adding child controls programmatically. |
| IsInPartialRendering | Indicates whether the panel is being updated as part of the partial post back. |
| RenderMode | Shows the render modes. The available modes are Block and Inline. |
| UpdateMode | Gets or sets the rendering mode by determining some conditions. |
| Triggers | Defines the collection trigger objects each corresponding to an event causing the panel to refresh automatically. |

### Methods of the UpdatePanel Control

The following table shows the methods of the update panel control:

|  |  |
| --- | --- |
| **Methods** | **Description** |
| CreateContentTemplateContainer | Creates a Control object that acts as a container for child controls that define the UpdatePanel control's content. |
| CreateControlCollection | Returns the collection of all controls that are contained in the UpdatePanel control. |
| Initialize | Initializes the UpdatePanel control trigger collection if partial-page rendering is enabled. |
| Update | Causes an update of the content of an UpdatePanel control. |

The behavior of the update panel depends upon the values of the UpdateMode property and ChildrenAsTriggers property.

|  |  |  |
| --- | --- | --- |
| **UpdateMode** | **ChildrenAsTriggers** | **Effect** |
| Always | False | Illegal parameters. |
| Always | True | UpdatePanel refreshes if whole page refreshes or a child control on it posts back. |
| Conditional | False | UpdatePanel refreshes if whole page refreshes or a triggering control outside it initiates a refresh. |
| Conditional | True | UpdatePanel refreshes if whole page refreshes or a child control on it posts back or a triggering control outside it initiates a refresh. |

## The UpdateProgress Control

The UpdateProgress control provides a sort of feedback on the browser while one or more update panel controls are being updated. For example, while a user logs in or waits for server response while performing some database oriented job.

It provides a visual acknowledgement like "Loading page...", indicating the work is in progress.

The syntax for the UpdateProgress control is:

<asp:UpdateProgress ID="UpdateProgress1" runat="server" DynamicLayout="true" AssociatedUpdatePanelID="UpdatePanel1" >

 <ProgressTemplate>

 Loading...

 </ProgressTemplate>

</asp:UpdateProgress>

The above snippet shows a simple message within the ProgressTemplate tag. However, it could be an image or other relevant controls. The UpdateProgress control displays for every asynchronous postback unless it is assigned to a single update panel using the AssociatedUpdatePanelID property.

### Properties of the UpdateProgress Control

The following table shows the properties of the update progress control:

|  |  |
| --- | --- |
| **Properties** | **Description** |
| AssociatedUpdatePanelID | Gets and sets the ID of the update panel with which this control is associated. |
| Attributes | Gets or sets the cascading style sheet (CSS) attributes of the UpdateProgress control. |
| DisplayAfter | Gets and sets the time in milliseconds after which the progress template is displayed. The default is 500. |
| DynamicLayout | Indicates whether the progress template is dynamically rendered. |
| ProgressTemplate | Indicates the template displayed during an asynchronous post back which takes more time than the DisplayAfter time. |

### Methods of the UpdateProgress Control

The following table shows the methods of the update progress control:

|  |  |
| --- | --- |
| **Methods** | **Description** |
| GetScriptDescriptors | Returns a list of components, behaviors, and client controls that are required for the UpdateProgress control's client functionality. |
| GetScriptReferences | Returns a list of client script library dependencies for the UpdateProgress control. |

## The Timer Control

The timer control is used to initiate the post back automatically. This could be done in two ways:

(1) Setting the Triggers property of the UpdatePanel control:

<Triggers>

 <asp:AsyncPostBackTrigger ControlID="btnpanel2" EventName="Click" />

</Triggers>

(2) Placing a timer control directly inside the UpdatePanel to act as a child control trigger. A single timer can be the trigger for multiple UpdatePanels.

<asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Always">

 <ContentTemplate>

 <asp:Timer ID="Timer1" runat="server" Interval="1000">

 </asp:Timer>

 <asp:Label ID="Label1" runat="server" Height="101px" style="width:304px" >

 </asp:Label>

 </ContentTemplate>

</asp:UpdatePanel>

# 5. C# ADO.NET SqlCommand - ExecuteNonQuery

The **ExecuteNonQuery()** is one of the most frequently used method in SqlCommand Object, and is used for executing statements that do not return result sets (ie. statements like insert data , update data etc.) .

 **Command.ExecuteNonQuery();**

The ExecuteNonQuery() performs Data Definition tasks as well as Data Manipulation tasks also. The Data Definition tasks like creating Stored Procedures ,Views etc. perform by the **ExecuteNonQuery()** . Also Data Manipulation tasks like Insert , Update , Delete etc. also perform by the ExecuteNonQuery() of SqlCommand Object.

The following C# example shows how to use the method ExecuteNonQuery() through SqlCommand Object.

using System;

using System.Windows.Forms;

using System.Data.SqlClient;

namespace WindowsApplication1

{

 public partial class Form1 : Form

 {

 public Form1()

 {

 InitializeComponent();

 }

 private void button1\_Click(object sender, EventArgs e)

 {

 string connetionString = null;

 SqlConnection cnn ;

 SqlCommand cmd ;

 string sql = null;

 connetionString = "Data Source=ServerName;Initial Catalog=DatabaseName;User ID=UserName;Password=Password";

 sql = "Your SQL Statemnt Here";

 cnn = new SqlConnection(connetionString);

 try

 {

 cnn.Open();

 cmd = new SqlCommand(sql, cnn);

 cmd.ExecuteNonQuery();

 cmd.Dispose();

 cnn.Close();

 MessageBox.Show (" ExecuteNonQuery in SqlCommand executed !!");

 }

 catch (Exception ex)

 {

 MessageBox.Show("Can not open connection ! ");

 }

 }

 }

}

Q.2 **DataReader**
DataReader as its name implies it is used to read the data. DataReader reads the data from database. DataReader gets the records from Stored Procedure , table returning SELECT query.
DataReader works only in forward direction means row read once cannot be read again because of this it is fast to fetching records.
Most used function of datareader is Read function which  reads the table row one by one and returns true or false if it is read.
Datareader always required a Open connection for executing the SQL commands. Once a connection closed then you will not been able to read the data from  datareader. Thats why it is used in connecte mode in SQL.

|  |
| --- |
| string conStr = ConfigurationManager.ConnectionStrings["constr"].ConnectionString;using (SqlConnection con = new SqlConnection(conStr)){    using (SqlCommand cmd = new SqlCommand("SELECT EmpId, EmpName FROM Employee", con))    {        cmd.CommandType = CommandType.Text;        con.Open();        SqlDataReader dr = cmd.ExecuteReader();        while (dr.Read())        {            string name = dr["EmpID"].ToString();            string city = dr["EmpName"].ToString();             Response.Write("EmpID: " + EmpID);                Response.Write("Name: " + EmpName);        }         con.Close();    }} |

**DataAdapter**
DataAdapter is also a ADO.Net object. It is used to execute the SQL statements and populate the result of that SQL query in DataSet or in DataTable.
DataAdapter gets all the returned rows from Sql statement at once and then fill the data into DataSet or datatable. Because of this dataadapter is slow in comparision to datareader.
DataReader have Read function just like Dataadapter has Fill function which fills the returned rows from sql statement onto dataset or datatable.
DataAdapter will not reuire any open and close connection . Means DataAdapter can work in Disconnected mode.

|  |
| --- |
| string constr = ConfigurationManager.ConnectionStrings["constr"].ConnectionString;using (SqlConnection con = new SqlConnection(constr)){    using (SqlCommand cmd = new SqlCommand("SELECT EmpId, EmpName FROM Employee", con))    {        cmd.CommandType = CommandType.Text;        using (SqlDataAdapter adp = new SqlDataAdapter(cmd))        {            DataSet ds = new DataSet();             adp.Fill(ds);            foreach (DataRow row in ds.Tables[0].Rows)            {                string EmpID = row["EmpID"].ToString();               string EmpName = row["EmpName"].ToString();                Response.Write("EmpID: " + EmpID);                Response.Write("Name: " + EmpName);            }        }    }} |

Q.2.2 There are two inputs, Username and word, and a login button. When the user clicks that login button the user is redirect to a new page (his account page), otherwise it gets an error message.

**INITIAL CHAMBER**
**Step 1**

Open Your Visual Studio 2010 and create an Empty Website, provide a suitable name (LoginForm\_demo).

**Step 2**

In Solution Explorer you get your empty website, then add two web forms and a SQL Server database as in the following.

**For Web Form**

Right-click LoginForm\_demo (your empty website) then select Add New Item -> Web Form. Name it Login\_demo.aspx. Use that process again and add another web form and name it Redirectpage.aspx.
 **For SQL Server Database**

Right-click LoginForm\_demo (your empty website) then select Add New Item -> SQL Server Database. Add the database inside the App\_Data\_folder.

**DATABASE CHAMBER**
**Step 3**

In Server Explorer, click on your database (Database.mdf) then select Tables -> Add New Table. Make the table like this:

Table -> tbl\_data (Don't forget to make the ID as IS Identity = True).


**Figure 1** Database Chamber

Now enter some data into your database by right-clicking Tables then select Show Table Data and enter whatever you want to add in the username and word. I entered the following data.


**Figure 2** Database Table

We will match this username and word so if the data is correct, the user is redirect to his account page otherwise the user gets an error message.

**DESIGN CODE**

**Step 4**

Now make some design for your application by going to Login\_demo.aspx and try the code as in the following.

**Login\_demo.aspx**

1. <body>
2. <form id="form1" runat="server">
3. <div >
4. <table style="width:100%;">
5. <caption class="style1">
6. <strong>Login Form</strong>
7. </caption>
8. <tr>
9. <td class="style2">
10. </td>
11. <td>
12. </td>
13. <td>
14. </td>
15. </tr>
16. <tr>
17. <td class="style2">
18. Username:</td>
19. <td>
20. <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
21. </td>
22. <td>
23. <asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server"
24. ControlToValidate="TextBox1" ErrorMessage="Please Enter Your Username"
25. ForeColor="Red"></asp:RequiredFieldValidator>
26. </td>
27. </tr>
28. <tr>
29. <td class="style2">
30. word:</td>
31. <td>
32. <asp:TextBox ID="TextBox2" TextMode="word" runat="server"></asp:TextBox>
33. </td>
34. <td>
35. <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
36. ControlToValidate="TextBox2" ErrorMessage="Please Enter Your word"
37. ForeColor="Red"></asp:RequiredFieldValidator>
38. </td>
39. </tr>
40. <tr>
41. <td class="style2">
42. </td>
43. <td>
44. </td>
45. <td>
46. </td>
47. </tr>
48. <tr>
49. <td class="style2">
50. </td>
51. <td>
52. <asp:Button ID="Button1" runat="server" Text="Log In" onclick="Button1\_Click" />
53. </td>
54. <td>
55. <asp:Label ID="Label1" runat="server"></asp:Label>
56. </td>
57. </tr>
58. </table>
59. </div>
60. </form>
61. </body>

You will get something like this:


**Figure 3** Login Form

**CODE CHAMBER**

**Step 5**

We will provide some code in the Login\_demo.aspx.cs page so that our login form works.

**Login\_demo.aspx.cs**


**Figure 4** Login Demo

1. public partial class \_Default: System.Web.UI.Page {
2. protected void Page\_Load(object sender, EventArgs e) {
3.
4. }
5. protected void Button1\_Click(object sender, EventArgs e) {
6. SqlConnection con = new SqlConnection(@
7. "Data Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\Database.mdf;Integrated Security=True;User Instance=True");
8. SqlCommand cmd = new SqlCommand("select \* from tbl\_data where username=@username and word=@word", con);
9. cmd.Parameters.AddWithValue("@username", TextBox1.Text);
10. cmd.Parameters.AddWithValue("word", TextBox2.Text);
11. SqlDataAdapter sda = new SqlDataAdapter(cmd);
12. DataTable dt = new DataTable();
13. sda.Fill(dt);
14. con.Open();
15. int i = cmd.ExecuteNonQuery();
16. con.Close();
17.
18. if (dt.Rows.Count > 0) {
19. Response.Redirect("Redirectform.aspx");
20. } else {
21. Label1.Text = "Your username and word is incorrect";
22. Label1.ForeColor = System.Drawing.Color.Red;
23.
24. }
25.
26.
27. }
28. }

**OUTPUT CHAMBER**

This is your final output. As you can see, this is our Default.aspx page (in your terms, Login\_demo.aspx) when the user clicks on the login button, the user is redirected to a new page (Redirectpage.aspx).



Q.3.1 **How to apply themes**There are 3 different options to apply themes to our website:

1. Setting the theme at the page level: the Theme attribute is added to the page directive of the page.

<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="Default" Theme="Theme1"%>
2. Setting the theme at the site level: to set the theme for the entire website you can set the theme in the web.config of the website. Open the web.config file and locate the <pages> element and add the theme attribute to it:

<pages theme="Theme1">
....
....
</pages>
3. Setting the theme programmatically at runtime: here the theme is set at runtime through coding. It should be applied earlier in the page's life cycle ie. Page\_PreInit event should be handled for setting the theme. The better option is to apply this to the Base page class of the site as every page in the site inherits from this class.

Page.Theme = Theme1;

**Uses of Themes**

1. Since themes can contain CSS files, images and skins, you can change colors, fonts, positioning and images simply by applying the desired themes.
2. You can have as many themes as you want and you can switch between them by setting a single attribute in the web.config file or an individual aspx page. Also you can switch between themes programmatically.
3. Setting the themes programmatically, you are offering your users a quick and easy way to change the page to their likings.
4. Themes allow you to improve the usability of your site by giving users with vision problems the option to select a high contrast theme with a large font size.

**Example**
Create 2 themes for the page â€“ one with red background (Theme1) and another with an image as a background (Theme2). When the user selects a particular theme from the ListBox then that theme should be applied dynamically for the page.



**Answer**

1. Solution Explorer -> Right click -> Add ASP.NET folder -> Themes.

A new folder App\_Themes is added to the Solution Explorer and a new folder Theme1 is added inside it.
2. Theme1 -> Right click -> Add new item -> Stylesheet -> name it as Theme1.css
3. Inside Theme1.css

body
{
      background-color:Red;
}
4. Again add a new file.

App\_Themes -> Right click -> Add ASP.NET folder -> Themes

A new folder Themes2 is created.
5. Themes2 -> Right click -> Add new item -> Stylesheet -> name it as Theme2.css
6. Create an Images folder inside Theme2 and add a picture file pic1.jpg inside this. File Theme2.css contains â€“

body
{
      background-image:url(Images/pic1.jpg);
}
7. Now create a default.aspx as follows. Add a heading and a list box having AutoPostBack to True.

<body>
    <form id="form1" runat="server">
    <div>
    <h3>Select your page Theme : </h3>
        <asp:ListBox ID="ListBox1" runat="server" AutoPostBack="True" Height="41px" onselectedindexchanged= "ListBox1\_SelectedIndexChanged" Width="175px">
                <asp:ListItem>Theme1</asp:ListItem>
                <asp:ListItem>Theme2</asp:ListItem>
            </asp:ListBox>
            <br />
      </div>
    </form>
</body>
8. Inside the default.aspx.cs file a static variable themeValue is defined which saves the value of current theme. In the Page\_PreInit event of the page, selected theme from the ListBox is applied to the page and inside the constrctor of the page this PreInit event is provided a EventHandler.

public partial class default : System.Web.UI.Page
{
  static string themeValue;

  public ThemeTest()
  {
    this.PreInit+=new EventHandler(Page\_PreInit);
  }

  private void Page\_PreInit(object sender, EventArgs e)
  {
    Page.Theme = themeValue;
  }

  protected void ListBox1\_SelectedIndexChanged(object sender, EventArgs e)
  {
    themeValue = ListBox1.SelectedItem.Value;
    Response.Redirect(Request.Url.ToString());
  }
}
9. Now run the page. When we select the Theme from the ListBox, immediately the page is automatically applied with the theme.

Q.3.2 Caching is one of the most interesting concepts and operations in ASP.NET. If you can handle it, you can run any web application by applying the caching concept depending on the requirements.

Currrently a majority of websites/portals (or I can say simply web pages) are dynamic (if I do talk about a dynamic website, then it doesn't mean all the pages of that website is dynamic or will be. The probability of happening this is dependent on the user's perspective and the requirements).

In veru common words I can define dynamic pages as including the following:

- Pages that directly interact with people
- Communication (on page)
- Any media content
- Any type of graphic interaction

So, generally these types of pages or webs are called dynamic. Now let's find why we really need caching.

**Why Caching**
Caching is for providing solutions or the results to the users depending on their requested request, admin needs to recreate the pages often depending on user requests…**STOP!!!

The process**
The process is quite bulky and time-consuming. So to overcoming that problem some websites have a page creation engine that automatically creates all the pages in one action and directly saves those pages as a HTML structured page. These HTML pages serve the user depending on their requirements.

**Multiple sorts of pages**

But, do you still think this will be enough? If your answer is yes, then please think some more!

Actually, the preceding solution will only work if and only if the requested pages are of the same type. Now think, what will happen if the users request a different sort of page?

In that case your web will be stuck again.

So for dealing with that kind of complex but necessary requirements, ASP.NET provides support for **caching**. Caching is the hero/heroine in this context that will help us to a great extent.
 **What a Cache does**

What a cache does, in the most simple words I can say is:

"A cache simply stores the output generated by a page in the memory and this saved output (cache) will serve us (users) in the future." That's it.

**Types**1Page Caching

2. Fragment Caching

3. Data Caching

**Page Caching**
Let's explore the caching of an entire page, first.

To cache an entire page's output we need to specify a directive at the top of our page, this directive is the @ OutputCache.

Let's figure out a simple demo of it.

1. <%@ OutputCache Duration = 5 VaryByParam = "ID" %>

Here, in that statement **Duration** and **VarByParam** are the two attributes of the **OutputCache** directive. Now let's explore how they work.

* **Duration Attribute**

This attributes represents the time in seconds of how long the output cache should be stored in memory. After the defined duration the content stored in the memory will be cleared automatically.
* **VarByParam Attribute**

This is the most important attributes; you can't afford to miss that in the OutputCache directory statement. It generally defines the query string parameters to vary the cache (in memory).

You can also multiple parameter names too, but for that you need to separate them using a semicolon (;).

You can also specify it as "\*". In this case the cached content is varied for all the parameters ed using the querysrting.

For example:

1. <%@ OutputCache Duration = 5 VaryByParam = "\*"%>

In case of caching a page, some pages can generate different content for different browsers. In that scenario we need to add an additional attribute to our statement for overcoming the preceding problem.

For example:

1. <%@ OutputCache Duration = 5 VaryByParam = "ID" VaryByCustom = "Browser" %>

Or:

1. <%@ OutputCache Duration = 5 VaryByParam = "\*" VaryByCustom = "Browser" %>

**Fragment caching**

In some scenarios we only need to cache only a segment of a page. For example a contact us page in a main page will be the same for all the users and for that there is no need to cache the entire page.

So for that we prefer to use fragment caching option.

For example:

1. <%@ OutputCache Duration = 10 VaryByParam = "None" %>

Or:

1. <%@ OutputCache Duration = 5 VaryByParam = "None" VaryByCustom = "Browser" %>

**Data Caching**
Data caching is slightly different from the 2 other caching types. It's much more interesting to see how data caching actually works.

As we know in C# everything is about classes and objects. So ASP.NET supports data caching by treating them as small sets of objects. We can store objects in memory very easily and use them depending on our functionality and needs, anywhere across the page.

Now you must be thinking **where is the class in that entire scenario**?

Actually, this feature is implemented using the Cache class and data is treated as its object. Let's see how it works using a demo.

I am inserting a string value in the cache as:

1. Cache["Website"] = "CSharpCorner";

Now, for inserting the cache into the objects, the insert method of the Cache class can be used. This insert method is used as follows:

1. Cache.Insert("Website", strName,
2. new CacheDependency(Sever.MapPath("Website.txt"));

**What we are missing something**
We missed the Time for the cache (don't forget to use it), let's provide it:

1. Cache.Insert("Website", strName,
2. new CacheDependency(Sever.MapPath("Website.txt")
3. DateTime.Now.Addminutes(5), TimeSpan.Zero);

Q.4.1 Globalization is the process of designing and developing applications that function for multiple cultures.

Localization is the process of customizing your application for a given culture and locale.

**Cultures and Locales**The language needs to be associated with the particular region where it is spoken, and this is done by using locale (language + location). For example: fr is the code for French language. fr-FR means French language in France. So, fr specifies only the language whereas fr-FR is the locale. Similarly, fr-CA defines another locale implying French language and culture in Canada. If we use only fr, it implies a neutral culture (i.e., location neutral).

**Set culture information**Application level -In web.config file<configuration>
  <system.web>
    <globalization culture="fr-FR" uiCulture="fr-FR"/>
  </system.web>
</configuration>



  **Resource Files**

1. A resource file is an XML file that contains the strings that you want to translate into different languages or paths to images.
2. The resource file contains key/value pairs. Each pair is an individual resource. Key names are not case sensitive.

**e.g.** A resource file might contain a resource with the key Button1 and the value Submit
Resource files in ASP. NET have an .resx extension. At run time, the .resx file is compiled into an assembly. **Global Resource Files**

1. You create a global resource file by putting it in the reserved folder App\_GlobalResources at the root of the application.
2. Any .resx file that is in the App\_GlobalResources folder has global scope.



**Local Resource Files**A local resources file is one that applies to only one ASP. NET page or user control (an ASP. NET file that has a file-name extension of .aspx, .ascx, or .master).

****

**Implicit Localization with Local Resources**

1. If you have created local resource files for a specific page, you can use implicit localization to fill the property values for a control from the resource file. In implicit localization, ASP.NET reads a resource file and matches resources to property values.
2. To use implicit localization, you must use a naming convention for resources in the local resource file that uses the  pattern:meta:resourcekey

**Explicit Localization**

1. Alternatively, you can use explicit localization, where you use a resource expression. Unlike implicit localization,   you must use a resource expression for each property that you want to set.<asp:Button ID="Button1" runat="server" Text="Button" Text="<%$ Resources:WebResources, Button1Caption %>" />

**Localizing Static Text**

1. If a page includes static text, you can use ASP.NET localization by including it in a Localize control, and then using explicit localization to set the static text. The Localize control renders no markup; its only function is to act as a placeholder for localized text.
2. The Localize control can be edited in Design view, not only in the property grid. At run time, ASP.NET treats the Localize controlas a Literal control.

      <asp:Localize ID="l1" runat="server" Text="Welcome!" meta:resourcekey="button1" />

**Get Resource value programmetically**Label3.Text = Resources.Resource.b1;
Label3.Text = GetLocalResourceObject("Button1.Text").ToString();
Label3.Text = GetGlobalResourceObject("resource", "color");

**Example

Create a Multi-Lingual Site with Localization

Web page**
 **Step 1**  Drag a button and Label
**Step 2** Right click on web site- Add ASP.NET Folder - Add App\_Local Resource -Select -
**Step 3** Right click on  Add New item - Resource File - Name (Default.aspx.resx)- Name (Button1.Text)- Ok / Label1.Text-English-Copy the RESX file and Paste it
**Step 4** Rename it - (Deafult.aspx.fr.resx)- edit it..(Button1.Text)- Namaste / Label1.Text- French.
 **.aspx page**<asp:Button meta:resourcekey="button1" />
<asp:Label meta:resourcekey="label1" />

**Header**<%@ Page Language="C#"  CodeFile="Default.aspx.cs" Inherits="\_Default" Culture="Auto:en-US" UICulture="Auto" %>

Next.. Right cilck on web site- Add ASP.NET Folder - Add App\_GlobalResources - Add Resource - Name and Value
copy and paste and rename (Resource.fr.resx)-

Now drag a new label :- label id(name)-renamecopypaste- expression - text - expression type=resource- expression property=class key=resource
Resource key-welcome

### Q.4.2 Session and Application variables basics

You can use the Application and Session objects to store values that are global rather than page-specific for either a particular user (the Session) or to all users (the Application).

The Session and Application variables are stored on the server. Client browsers are then attached to the session through a cookie. As a result, the client must have cookies enabled in the browser for Session and Application variables to work.

The samples to follow demonstrate how to use Visual InterDev to create Session and Application variables.

### Create a Visual InterDev Web project

1. Create a new Web Project in Visual InterDev. For detailed instructions, see the following article in the Microsoft Knowledge Base:

[301184](https://support.microsoft.com/en-us/help/301184) How to create a Visual InterDev project

1. From the Project menu, point to **Add Web Item**, and then click
**Active Server Page**.
2. Click Open to accept the default information for the new ASP page.

Now that you added an ASP page to the current project, the ASP code samples in the next two sections demonstrate the use of both Session and Application variables.

### How to use Session variables

The power of the Session object comes from the fact that it can store variables that are global to just that specific user; as a result, each user can have their own individual value for that variable. Session objects are not always created automatically for every user when they enter your application. However, storing or accessing a variable in the Session object creates the Session object and fires the Session\_OnStart event.

To demonstrate how to use the Session object in an ASP page, follow these steps:

1. Paste the following code between the <BODY> </BODY> tags of the ASP page that you created earlier in Visual InterDev:

<%
 'Store information in a Session variable.
 Session("myInformation") = "somevalue"

 'Display the contents of the Session variable.
 Response.Write Session("myInformation")
%>

1. Click **View in Browser** from the View menu.
2. When you are prompted to save the file, click OK. The browser displays the information in the variable.

### How to use Application variables

Think of the Application object as a global container for information that is available to all pages of your ASP application. You can store variables and object references in the Application object. The Application object is instantiated when the first page of your application is requested and remains available until the Web service is shut down.

To demonstrate how to use the Application object in an ASP page, follow these steps:

1. Paste the following code between the <BODY> </BODY> tags of the ASP page that you created earlier in Visual InterDev:

<%
 'Store information in an Application variable.
 Application("myvalue") = "something"

 'Display the contents of the Application variable.
 Response.Write Application("myvalue")
%>

1. Click **View in Browser** from the View menu.
2. When you are prompted to save the file, click OK. The browser displays the information in the variable.

### Application\_OnStart and Session\_OnStart events

You can use both the Application\_OnStart and Session\_OnStart events to initialize state. To do this, set up variables that are global for either the application or a specific user. When the first user accesses a file in your application, the Application\_OnStart event is triggered. This event is used to initialize any application-wide global variables. When the user begins a session for the first time, the Session\_OnStart event is triggered. This event is used to initialize user-specific information.

### Application\_OnEnd and Session\_OnEnd events

The Session\_OnEnd event occurs either when a current session times out. By default, this is 20 minutes after the last request for a page from the application. To change this default time out setting, you can set the Session.Timeout property or edit the registry.

You can use the Application\_OnEnd event to clean up all of the global objects and variables. However, in a present problem, this event may not be triggered until the Web server is stopped. Revisions of ASP seem likely to specify that the Application\_OnEnd event be triggered once the last Session\_OnEnd event occurs, that is, when the last session ends and there are no current application users.

### Troubleshooting

Application and Session variables require that clients have cookies enabled in their Web browser, which is not reliable between shared frames in a frameset or across multiple browser instances.

Session and Application events are only triggered when a client retrieves an ASP page; they are not triggered when an HTML page in the application is requested. Therefore, if you have additional server-side applications such as Internet Server Application Programming Interface (ISAPI) or Common Gateway Interface (CGI) scripts, make sure that these applications do not depend on specific events having occurred within an ASP page. Otherwise, the ISAPI or CGI script may fail (crash) and cause the Web server to stop responding (hang).

This method stores information on the Web server. This decreases scalability in two ways: Session and Application variables use resources on the server and cannot be used on completely load balanced Web Farms.